



**VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ**  
BRNO UNIVERSITY OF TECHNOLOGY

**FAKULTA INFORMAČNÍCH TECHNOLOGIÍ**  
FACULTY OF INFORMATION TECHNOLOGY

**ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ**  
DEPARTMENT OF COMPUTER SYSTEMS

**EMULÁTOR HERNÍ KONZOLE PLAYSTATION 1 S VYŠŠÍM RENDEROVACÍM ROZLIŠENÍM**

PLAYSTATION 1 EMULATOR WITH HIGHER RENDERING RESOLUTION

**SEMESTRÁLNÍ PROJEKT**  
TERM PROJECT

**AUTOR PRÁCE**  
AUTHOR

**Bc. FILIP STUPKA**

**VEDOUCÍ PRÁCE**  
SUPERVISOR

**doc. Ing. JIŘÍ JAROŠ, Ph.D.**

**BRNO 2024**

## Abstrakt

Tato práce se zabývá zvýšením vizuální věrnosti v emulaci systému PlayStation 1 zavedením pokročilých vykreslovacích technik pro výstup ve vyšším rozlišení. Studie začíná důkladnou analýzou původní hardware architektury a identifikací komponent ovlivňujících grafický výstup. Výzkum se zaměřuje na vývoj vlastního emulátoru a zabývá se problémy spojenými se zvýšením rozlišení grafiky při zachování kompatibility se stávajícím herním softwarem. Výsledky poskytují cenné poznatky pro nadšence retro her a výzkumné pracovníky a ukažují úspěšnou implementaci emulátoru PlayStation 1 s vyšším rozlišením vykreslování pro zachování odkazu klasických her.

## Abstract

This thesis investigates the augmentation of visual fidelity in PlayStation 1 emulation by implementing advanced rendering techniques for higher resolution output. The study begins with a thorough analysis of the original hardware architecture, identifying components impacting graphical output. Focusing on developing a custom emulator, the research addresses challenges associated with upscaling graphics while maintaining compatibility with existing game software. The findings provide valuable insights for retro gaming enthusiasts and researchers, showcasing the successful implementation of a PlayStation 1 emulator with higher rendering resolution to preserve the legacy of classic games.

## Klíčová slova

emulátor, PlayStation, Sony, simulátor, grafika, rozlišení

## Keywords

emulator, PlayStation, Sony, simulator, graphics, resolution

## Citace

STUPKA, Filip. *Emulátor herní konzole Playstation 1 s vyšším renderovacím rozlišením*. Brno, 2024. Semestrální projekt. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Jiří Jaroš, Ph.D.

# Emulátor herní konzole Playstation 1 s vyšším renderovacím rozlišením

## Prohlášení

Prohlašuji, že jsem tuto semestrální práci vypracoval samostatně pod vedením pana doc. Ing. Jiřím Jarošem, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....  
Filip Stupka  
22. ledna 2024

## Poděkování

Děkuji svému vedoucímu doc. Ing. Jiřímu Jarošovi, Ph.D. za vedení této práce a věcné připomínky.

# Obsah

<b>1</b>	<b>Úvod</b>	<b>4</b>
1.0.1	Dostupnost her . . . . .	4
1.0.2	Emulátory . . . . .	4
<b>2</b>	<b>PlayStation</b>	<b>6</b>
2.1	NTSC/PAL a bezpečnost . . . . .	6
2.2	Existující emulátory . . . . .	8
2.3	Renderovací rozlišení . . . . .	9
<b>3</b>	<b>Architektura</b>	<b>11</b>
3.1	Nástroje . . . . .	11
3.2	Hardwarová komponenta . . . . .	11
3.3	Návrh architektury . . . . .	12
<b>4</b>	<b>Hardware komponenty</b>	<b>14</b>
4.1	Sběrnice . . . . .	14
4.1.1	Virtuální paměť . . . . .	15
4.1.2	Vyrovnávací paměť instrukcí . . . . .	16
4.1.3	Časování . . . . .	16
4.2	CPU . . . . .	16
4.2.1	Vnitřní stav <i>CPU</i> . . . . .	17
4.2.2	Zpoždění načítání hodnot . . . . .	17
4.2.3	Zpoždění skoku . . . . .	17
4.2.4	Instrukční sada . . . . .	17
4.2.5	Koprocesory . . . . .	18
4.3	GPU . . . . .	19
4.3.1	VRAM . . . . .	19
4.3.2	Geometrická primitiva . . . . .	20
4.3.3	Barvy . . . . .	20
4.3.4	Rasterizace primitiv . . . . .	21
4.3.5	Zvýšení rozlišení . . . . .	21
4.4	DMA . . . . .	21
4.5	Ovladač přerušení/výjimek . . . . .	22
4.6	Časovače . . . . .	23
4.7	MDEC . . . . .	23
4.8	CD-ROM mechanika . . . . .	24
4.9	Periférie . . . . .	24

<b>5 BIOS</b>	<b>25</b>
5.1 Odrazový bod . . . . .	25
5.2 Funkce . . . . .	25
5.3 Fáze BIOSu . . . . .	26
5.3.1 Zaváděcí fáze . . . . .	26
5.3.2 Jádro BIOSu . . . . .	26
<b>6 Závěr</b>	<b>28</b>
<b>Literatura</b>	<b>29</b>

# Seznam obrázků

2.1	První pokus společnosti <i>Sony</i> proniknout na herní trh byl velmi úspěšný. Celkem se prodalo přibližně 102,49 milionu kusů. Pro srovnání, <i>Nintendo 64</i> od společnosti <i>Nintendo</i> se prodalo pouze 32,93 milionu . . . . .	7
2.2	Nedávno se objevil prototyp konzole, která je důkazem vztahu mezi společnostmi <i>Sony</i> a <i>Nintendo</i> . Konzole fungovala jako hybrid, do kterého bylo možné nahrávat hry z <i>CD</i> nebo kazet. Nakonec tato konzole nebyla nikdy vydána, protože <i>Sony</i> chtělo kontrolovat licencování <i>CD</i> verzí her a <i>Nintendo</i> to tento požadavek zamítlo[2]. . . . .	8
2.3	<i>Dolphin</i> emulátor umí zvýšit interní renderovací rozlišení, což může vylepsit herní zážitek . . . . .	9
3.1	Rozhraní, které každá hardwarová komponenta musí respektovat. . . . .	12
3.2	Kompletní návrh zapojení nejen hardwarových komponent emulátoru, ale i nadstavby nutné pro zpřístupnění stavu konzole. . . . .	13
4.1	Sběrnice pomocí <i>shared_ptr</i> struktury může vlastnit jednotlivé komponenty, ale také umožňuje sdílení mezi těmito komponentami. . . . .	14
4.2	Kompletní mapa všech <b>69</b> instrukcí podle specifikace[7]. . . . .	18
4.3	<i>MIPS</i> má celkem 3 způsoby, jak zakódovat instrukci do 32 bitů. . . . .	19
4.4	Pomocí 7 instrukcí a správným prokládáním lze docílit maximální rychlosti při kopírování dat. Každý cyklus přenese 4 byty. . . . .	21
4.5	<i>MDEC</i> má velmi podobnou strukturu jako <i>JPEG</i> až na vynechaný krok Huffmanova dekódování. . . . .	24
5.1	Úvodní obrazovka indikující správnou inicializi systému. . . . .	27

# Kapitola 1

## Úvod

Videoherní průmysl, ačkoliv je stále relativně mladý, představuje jednu z nejvýdělečnějších a nejdůležitějších forem médií v moderní době. Nejenže se stále více integruje do kultury, ale již v roce 2018 byl tento průmysl schopen vygenerovat 134,9 miliardy dolarů<sup>[5]</sup>, a toto číslo každým rokem stále roste.

S tímto růstem a úspěchem však přichází určitá forma amnézie. Většina firem, které se zabývají tvorbou a publikováním videoher, se soustředí pouze na moderní systémy. Ty hry, které byly v minulosti odpovědné za vývoj a růst tohoto průmyslu, rychle mizí z dějin lidské kultury, podobně jako pára nad hrncem.

### 1.0.1 Dostupnost her

V roce 2023 byla provedena studie<sup>[8]</sup> společností *Video Game History Foundation*, zaměřující se na dostupnost starých videoher na moderních systémech, a to zda-li je vůbec možné zakoupit staré videohry oficiálně a legální cestou.

Výsledek tohoto výzkumu byl skličující. Z 4000 vtipovaných videoher vydaných před rokem 2010 je nedostupných **87%**<sup>[8]</sup>, a není možné je získat oficiální legální cestou (studie se zabývala pouze obchody v USA). Článek tvrdí, že existuje několik důvodů, proč je historie videoherního průmyslu v takto zanedbaném stavu.

Jedním z důvodů jsou technické problémy, kdy portování videohry z jednoho systému na jiný může být netriviální záležitost a vyžaduje alokaci finančních zdrojů. Dalším a důležitějším faktorem nedostupnosti videoher je problém licenčních práv, kde samotný zákon ztěžuje dostupnost a samozřejmě také distribuční prostředky mohou v tomto problému hrát roli.

I přesto, jak je tento problém široce pochopen, není mu přisuzována téměř žádná váha v širších kruzích.

### 1.0.2 Emulátory

Většina způsobů, jak získat možnost hrát historické videohry, ve většině případů hraničí se zákonem. Člověk je nucen nelegálně stahovat software z internetových stránek, které jsou dedikované pro protizákonému sdílení licencovaného obsahu.

Ovšem to je možné pouze tehdy, pokud daná videohra byla vytvořena pro hardwarový systém, který daný člověk již vlastní. Jestliže byla videohra publikována pouze na platformě, která se již 20 let neprodává a není již nijak oficiálně podporována, má člověk smůlu. To

platí i tehdy když si člověk oficiálně hru zakoupil, ale poté nemá dostupný hardware, který by hru dokázal spustit.

Problém nedostupnosti samotných her je obtížně řešitelný, avšak problém nedostupnosti historického hardwaru je na tom podstatně lépe. Ačkoliv herní konzole jsou speciálně navrženy a tedy patentovány, nic veřejnosti nebrání vytvořit softwarový simulátor, který se snaží co nejpřesněji napodobit chování hardwaru herních konzolí a zároveň umožnit určitou videohru si zahrát, jak ukázal soudní spor mezi firmami Sony Computer Entertainment a Connectix Corporation[1]. Implementace emulátoru jedné staré herní konzole je náplní této práce.

## Kapitola 2

# PlayStation

V roce 1995 se společnost *Sony Computer Entertainment* rozhodla vydat herní konzoli *PlayStation*. Toto rozhodnutí bylo reakcí na rozpadlý vztah s firmou *Nintendo*, se kterou měla spolupracovat na vytvoření nové herní konzole, umožňující využívání jak *CD*, tak i kazet jako úložné médium. Díky ambicím ze strany *Sony* a nepřátelství vůči *Nintendu* se *PlayStation* konzole odlišovala v mnoha ohledech od svých konkurentů, a právě tyto změny vyvolaly významný ohlas a zajistila této konzoli úspěch.

V současné době je *PlayStation* šestou nejlépe prodávanou herní konzolí<sup>1</sup>, čímž se stala velmi populárním systémem. Konzole obsahovala následující hardware<sup>2</sup>:

- **CPU** - MIPS R3000A 33.8688MHz
- **RAM** - 2MiB EDO DRAM
- **Geometry Transformation Engine (GTE)** - Akcelerátor lineární algebry
- **Motion Decoder (MDEC)** - Dekodér JPEG obrázků
- **GPU** - 32bit Sony GPU, 1MB VRAM
- **SPU** - 16bit Sony SPU
- **CD-ROM**

### 2.1 NTSC/PAL a bezpečnost

V roce 1995 byla hlavní televizní technologií stále *Cathode Ray Tube (CRT)*. Nicméně existovaly dva standardy, které specifikovaly enkódování a zobrazování barev na těchto analogových zařízeních. Tyto standardy byly pojmenovány **NTSC** a **PAL**, přičemž to s jakým standardem se člověk mohl setkat záviselo na geografické poloze. *PlayStation* konzole samozřejmě musela s těmito rozdíly počítat a rozlišovala celkem tři různé regiony:

- **USA** - NTSC

---

<sup>1</sup>Seznam nejlépe prodávaných videoherních konzolí [https://en.wikipedia.org/wiki/List\\_of\\_best-selling\\_game\\_consoles](https://en.wikipedia.org/wiki/List_of_best-selling_game_consoles)

<sup>2</sup>Technická specifikace PlayStation konzole [https://en.wikipedia.org/wiki/PlayStation\\_technical\\_specifications](https://en.wikipedia.org/wiki/PlayStation_technical_specifications)



Obrázek 2.1: První pokus společnosti *Sony* proniknout na herní trh byl velmi úspěšný. Celkem se prodalo přibližně 102,49 milionu kusů. Pro srovnání, *Nintendo 64* od společnosti *Nintendo* se prodalo pouze 32,93 milionu

- **Japonsko - PAL**
- **Evropa - PAL**

Tyto dva standardy definovaly vertikální rozlišení, snímkovou frekvenci a také verzi *BIOSu*, který konzole obsahovala<sup>3</sup>. Aby se *Sony* vyhnulo licenčním problémům a předešlo pirátství, regionálně uzavřelo každou konzoli tak, že každá konzole měla ve svém *BIOSu* regionální podpis. Poté na každém *CD* obsahujícím hru byla vypálena jedna ze tří regionálních značek. Tento speciální segment se nacházel velmi blízko středového otvoru *CD* a konvenční *CD-ROM* čtečky nebyly schopné tento region číst ani do něj cokoliv vypalovat.

Při bootování hry byl tento speciální segment přečten a porovnán s regionálním kódem *BIOSu*, a pokud se tyto podpisy neshodovaly, konzole odmítla hru načíst.

Tento způsob ochrany byl velmi prostý, ale účinný. Speciální vypalovače vlastnila pouze *Sony*, a tedy oficiálně licencované *CD* disky nebylo možné perfektně zreplikovat. Nicméně tato forma ochrany nebyla perfektní a relativně brzy byly objeveny způsoby, jak tento systém obejít.

Prvním způsobem bylo vytvoření hardwarového čipu, který člověk nainstaloval do konzole<sup>4</sup>. Tento čip pak figuroval jako prostředník mezi *CD-ROM* a *BIOSem* a kdykoliv byl speciální region vyžádán *BIOSem*, čip zaslal falešná data s nimiž byl *BIOS* spokojen.

Druhým způsobem obejití ochrany byla technika nazvaná *Disc Swapping*. Tento exploit byl založen na faktu, že *CD-ROM* mechanika používala fyzikální spínač, pomocí kterého detekovala, zda je poklop zavřen či otevřen. Uživatel mohl tento spínač, například pomocí kusu plastu nebo složeného papíru, uměle přinutit si myslet, že poklop je uzavřen. Uzavření poklopu byl signál pro konzoli, aby začala načítat hru a provést kontrolu regionálního kódu. Jakmile kontrola byla provedena, uživatel se stále otevřeným poklopem oficiální *CD* rychle vyměnil za neoficiální *CD*, a tím byl exploit hotov. Konzola pak si myslela, že obsahuje legitimní oficiální *CD*, což ale nebyla pravda.

<sup>3</sup>NTSC/PAL rozdíly[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#vertical-video-timings>

<sup>4</sup>Stealth chip <https://www.r43ds.org/products/PS1-Modchip-Playstation-Stealth-Mod-chip.html>



Obrázek 2.2: Nedávno se objevil prototyp konzole, která je důkazem vztahu mezi společnostmi *Sony* a *Nintendo*. Konzole fungovala jako hybrid, do kterého bylo možné nahrávat hry z *CD* nebo kazet. Nakonec tato konzole nebyla nikdy vydána, protože *Sony* chtělo kontrolovat licencování *CD* verzí her a *Nintendo* to tento požadavek zamítlo[2].

## 2.2 Existující emulátory

Popularita této konzole je samozřejmě patrná z mnoha hledisek. Nejenže v roce 2020 *Sony* vydalo pátou generaci své herní konzole, ale dodnes existuje dedikovaná komunita nadšenců, kteří tuto konzoli dokázali rozebrat do posledního šroubku[4]. Existují rozsáhlé dokumenty, které popisují tuto konzoli do každého detailu, a díky tomu existuje celá řada emulátorů. To, co také činí tento systém populárním v těchto kruzích je, že na tehdejší dobu tato konzole měla z velké části modularní design. Herní systémy té a předchozí doby měly mnoho komponent, které byly doslova *natvrdo zadrátovány* a neumožňovaly žádnou flexibilitu. I přesto, že *PlayStation* do této kategorie hardwarového designu částečně spadá, celý systém je navržen spíše jako moderní stolní počítač.

V dnešní době emulátory nejen poskytují velmi přesnou emulaci *PlayStation* systému, ale také nabízejí nespouštěcí vychytávek oproti původnímu systému, které zpříjemňují jeho použití. Například emulátor *DuckStation*<sup>5</sup> umožňuje uložit současný stav celé konzole (*Save State*) a vytvořit tak snímek v čase, který člověk může později obnovit. Tato vlastnost se hodí v obtížných videohrách, ve kterých se ukládá postup zřídka, a tedy umožňuje hráči si diktovat uložení postupu dle vlastního uvážení.

Další vymožeností, kterou disponuje emulátor *PCSX ReARMed*<sup>6</sup>, je vlastní implementace *High-Level Emulation (HLE)* *BIOSu*. *BIOS* je nedílnou součástí každé *PlayStation* konzole. Stará se jednak o inicializaci hardwaru, ale také figuruje jako velmi jednoduchý operační systém, který videohry, stavěné na tomto systému, mohou používat pro usnadněný přístup k hardwaru. Tento fakt nutí každého uživatele emulátoru *BIOS* extraovat z původní konzole nebo jej získat nelegálními prostředky. Emulátor *PCSX ReARMed* se pokusil celý *BIOS* dekomplikovat a naimplementovat ho přímo do vlastního systému.

<sup>5</sup>Duckstation github repozitář <https://github.com/stenzek/duckstation>

<sup>6</sup>PCSX ReARMed github repozitář [https://github.com/notaz/pcsx\\_rearmmed](https://github.com/notaz/pcsx_rearmmed)

Většina emulátorů také nabízí možnost *Just-in-Time (JIT)* komplikace *PlayStation* procesoru buď na *x86\_64* či *arm* architekturu. Transformace instrukcí *PlayStation* procesoru na nativní procesor za běhu má za následek velké zisky v oblasti výkonu. Emulátor *PCSX ReARMed* jde ještě dál a určité hardwarové komponenty *PlayStation* systému implementuje pomocí ručně psaného *arm assembly* kódu, což umožňuje získat nemálo výkonu zejména u mobilních zařízení, u kterých dominuje *arm* architektura.

Velmi impozantním projektem je také *MiSTer FPGA PSX*<sup>7</sup>, což je práce snažící se implementovat *PlayStation* emulátor na *FPGA* čipu.

## 2.3 Renderovací rozlišení

Nepochybně je ekosystém okolo konzole *PlayStation* velmi vyzrálý. Mít možnost si bez větších problémů užít videohry z 90. let je velmi uspokojivé z hlediska zachování videoher historie. Nicméně i přesto že existuje několik přesných emulátorů s mnoha vychytávkami, žádný z nich se nesoustřídí na vylepšení grafické prezentace, jako například možnost změnit velikost interního renderovacího rozlišení<sup>8</sup>. Tento nápad není originální a lze ho nalézt v emulátorech různých konzolí jako jsou například:

- **Citra** - emulátor *Nintendo 3DS*<sup>9</sup>
- **Dolphin** - emulátor *Nintendo Gamecube/Wii*<sup>10</sup>
- **PCSX2** - emulátor *Sony PlayStation 2*<sup>11</sup>



Obrázek 2.3: *Dolphin* emulátor umí zvýšit interní renderovací rozlišení, což může vylepšit herní zážitek

Tato schopnost emulátoru je možná díky tomu, že systémy jako *Nintendo Gamecube* či *PlayStation 2* mají podobnou grafickou pipeline jako moderní systémy, a tedy není tak

<sup>7</sup>MiSTer FPGE PSX github repozitář [https://github.com/MiSTER-devel/PSX\\_MiSTER](https://github.com/MiSTER-devel/PSX_MiSTER)

<sup>8</sup>Ukázka změny rozlišení <https://www.youtube.com/watch?v=LSIYalc1D6Y>

<sup>9</sup>Citra github repozitář <https://github.com/citra-emu/citra>

<sup>10</sup>Citra github repozitář <https://github.com/dolphin-emu/dolphin>

<sup>11</sup>Citra github repozitář <https://github.com/PCSX2/pcsx2>

složité modifikovat emulátor tak, aby vykreslovat do většího prostoru. *PlayStation* má velmi netradiční zpracování grafiky (např.: chybějící *z-buffer*, žádná perspektivní korekce, ...) a vykreslování ve vyšším rozlišení je tedy složitější, protože nemálo her na těchto netradičních hardwarových funkcích napevno závisí. Tato práce se tedy nezabývá pouze implementací emulátoru, ale také zvyšováním renderovacího rozlišení *GPU* komponenty systému.

# Kapitola 3

# Architektura

Emulátor jakéhokoliv hardwarového systému je ve své podstatě velmi komplexní, proto je třeba správně rozvrhnout celkovou implementaci tohoto projektu. Zaprve, je nezbytné myslit na udržitelnost kódu a schopnost jej bez větších problémů rozšiřovat. Za druhé, komplexita kódu by měla být rozdělena do jednotlivých jednotek. Emulátor také potřebuje přístup k oknu a musí být schopen vykreslovat obsah grafické paměti na obrazovku.

## 3.1 Nástroje

Pro tyto účely a požadavky jsem zvolil *C++* jako jazyk pro implementaci. *Objektivně orientovaný* přístup k designu projektu a *polymorfismus* tohoto jazyka jsou pro tento projekt ideální.

Pro přístup ke grafickému rozhraní jsem zvolil knihovnu *SDL2*, což je univerzální nástroj pro otevření oken v operačním systému a možnost kreslení do nich. *SDL2* mimo jiné také umožňuje práci s obrázky a hudbou.

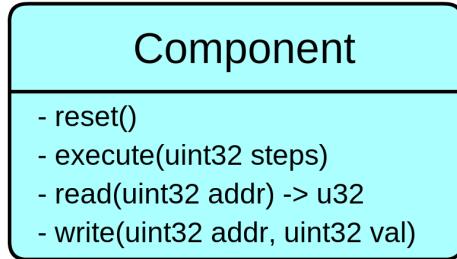
## 3.2 Hardwarová komponenta

*PlayStation* ve svém hardwarovém designu připomíná velice obyčejný počítač, kterému byly odstraněny přebytečné komponenty. *Sony*, na rozdíl od svých oponentů, vytvořilo tuto konzoli z relativně dobře dokumentovaných čipů již existujících počítačů. Samozřejmě *PlayStation* obsahuje i patentované, na míru udělané, součástky, které se snaží ulehčit práci procesoru.

Tyto hardwarové čipy sdílejí velmi podobné rozhraní. To je dáné tím, že *PlayStation* je nadesignován jako *Memory-mapped I/O*. To znamená, že sběrnice systému má uniformní paměťové rozložení, přičemž jednotlivé komponenty se mapují do specifických paměťových intervalů. Jednotlivé komponenty pak mohou do těchto intervalů zapisovat nebo číst a sběrnice pak rozdistribuuje tyto přístupy daným komponentám. Díky tomuto faktu musí mít každá hardwarová komponenta následující schopnosti:

- *Reset/Inicializace* komponenty
- *Čtení* z komponenty
- *Zápis* do komponenty
- Provedení *jednotky práce* dané komponenty

Pomocí dědičnosti a virtuálních metod v *C++* můžeme specifikovat virtuální třídu, která bude sloužit jako základ pro všechny hlavní hardwarové komponenty.



Obrázek 3.1: Rozhraní, které každá hardwarová komponenta musí respektovat.

### 3.3 Návrh architektury

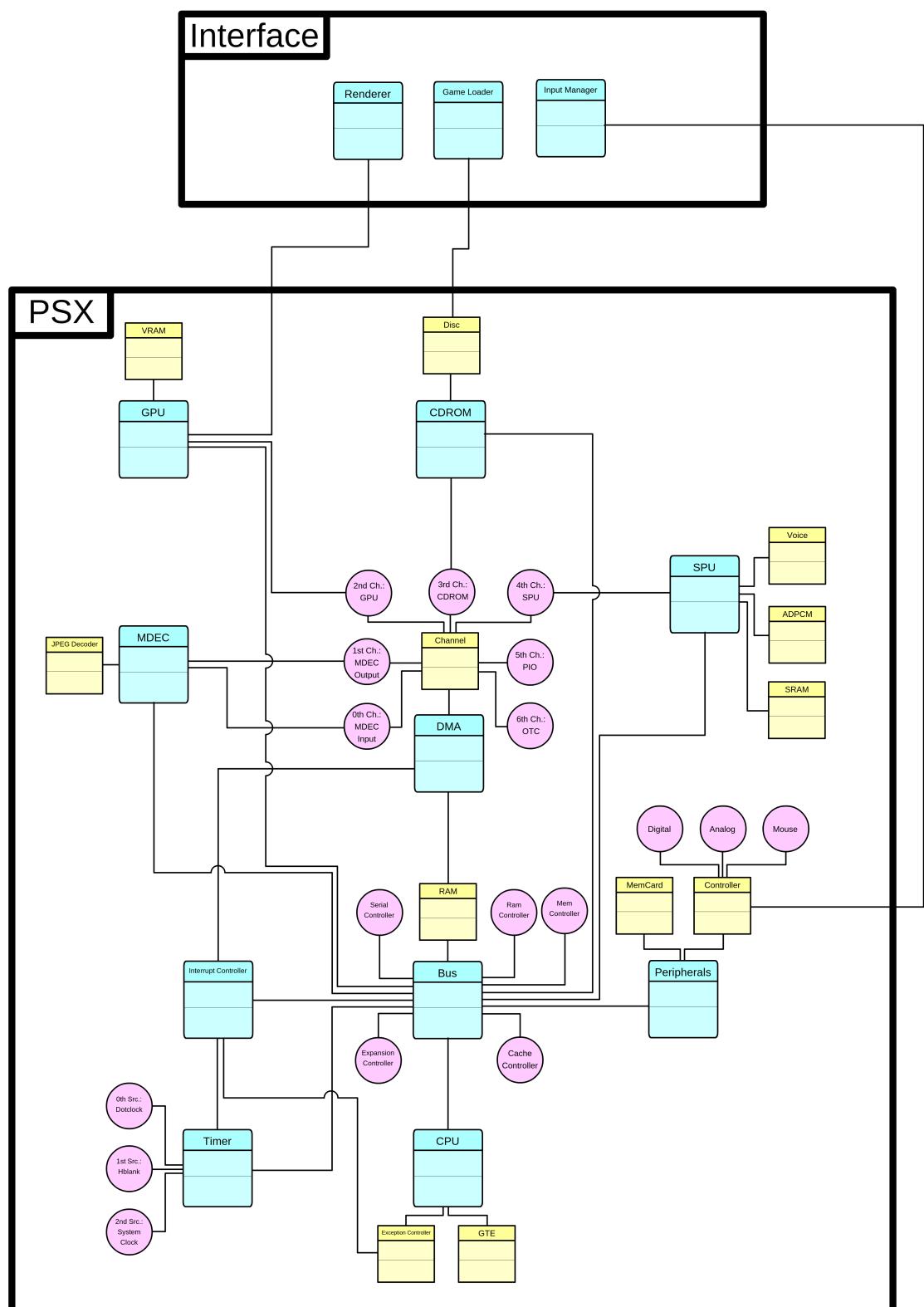
*PlayStation* se skládá z několika hlavních hardwarových komponent. Patří sem[4]:

- Sběrnice
- Central Processing Unit (**CPU**)
- Graphics Processing Unit (**GPU**)
- Sound Processing Unit (**SPU**)
- 3 Časovače/Hodiny
- Ovladač přerušení
- Ovladač přímého přístupu do paměti (**DMA**)
- Dekodér makrobloku (**MDEC**)
- CD-ROM

Každá z těchto komponent je klíčová pro správné fungování emulátoru jako celku. Detaily propojení komponent a schopností, s kým mohou komunikovat, jsou popsány v celkovém návrhu[3.2].

V návrhu jsou také zohledněny podřadné komponenty, které nemohou fungovat samostatně. To se týká například komponenty *Geometry Transformation Engine (GTE)*, což je ko-procesor specializující se na práci s lineární algebrou. Jelikož k této komponentě lze přistoupit pouze skrze *CPU* pomocí speciální instrukce, nelze tuto komponentu chápat jako samostatný celek.

Každá komponenta je následně propojena se sběrnicí kvůli *Memory-mapped I/O*. Existují však i přímá propojení bez sběrnice jako prostředníka. To je hlavně díky *DMA* komponentě, která zajistuje rychlý přenos dat, aniž by se *CPU* muselo starat o tento přenos. Další přímá propojení jsou určena pro správu přerušení. Vzhledem k tomu, že přerušení může nastat skoro v každé komponentě, je nutné toto přerušení přenášet do *Ovladače vyjímek*, který následně upraví stav *CPU* a přerušení se zpracuje jako výjimka.



Obrázek 3.2: Kompletní návrh zapojení nejen hardwarových komponent emulátoru, ale i nadstavby nutné pro zpřístupnění stavu konzole.

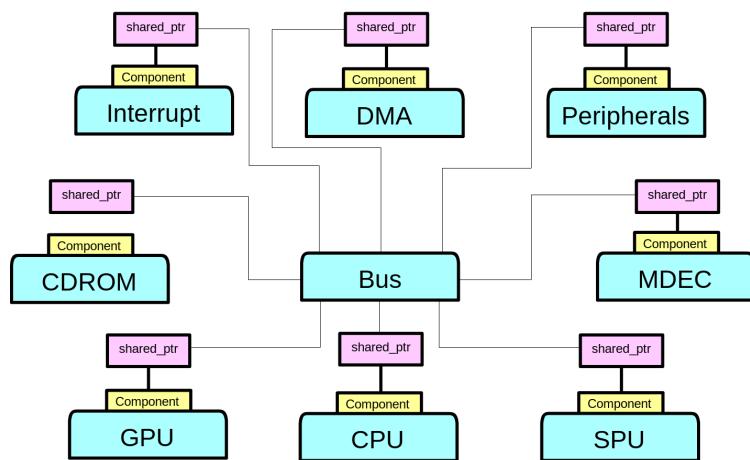
# Kapitola 4

## Hardware komponenty

### 4.1 Sběrnice

Sběrnice představuje do jisté míry nervový systém celého emulátoru, neboť nejen propojuje hlavní hardwarové komponenty, ale musí se také starat o jejich paměťovou správu. Díky tomu faktoru, sběrnice v emulátoru vytváří všechny komponenty a řeší jejich vzájemné závislosti.

Jelikož komponenty mohou být vzájemně sdíleny, bylo nutné zvolit vhodnou strukturu pro správu jednotlivých objektů. Pro tyto účely jsem se rozhodl použít `shared_ptr` ze standardní knihovny C++. Nejenže paměť bude spravována automaticky, ale lze velice jednoduše vytvořit duplicitní reference na tentýž objekt.



Obrázek 4.1: Sběrnice pomocí `shared_ptr` struktury může vlastnit jednotlivé komponenty, ale také umožňuje sdílení mezi těmito komponentami.

Další zodpovědností sběrnice je správné distribuce čtení a zápisů jednotlivým komponentám. Všechny tyto paměťové operace závisejí na *Memory-mapped I/O*.

*I/O* mapuje lineární 32-bitový paměťový prostor na segmenty, skrz které může sběrnice ovládat jednotlivé komponenty<sup>1</sup>.

Tabulka 4.1: Paměťová mapa

Název	Lokace	Velikost
RAM	0x00000000	2 MiB
Expansion	0x1F000000	1 MiB
Scratchpad	0x1F800000	1 KiB
Ovladač paměti	0x1F801000	36 B
Periférie	0x1F801040	16 B
Serial	0x1F801050	16 B
Ovladač RAM	0x1F801060	4 B
Ovladač přerušení	0x1F801070	8 B
DMA	0x1F801080	128 B
DotClock Časovač	0x1F801100	16 B
HBlank Časovač	0x1F801110	16 B
SystemClock/8 Časovač	0x1F801120	16 B
CDROM	0x1F801800	4 B
GPU	0x1F801810	8 B
MDEC	0x1F801820	8 B
SPU	0x1F801C00	1 KiB
I/O porty	0x1F802000	8 KiB
BIOS	0x1FC00000	512 KiB
Ovladač cache	0x1FFE0130	4 B

Jelikož sběrnice má 32-bitovou šířku, mohlo by se zdát, že stačí pouze na základě paměťové mapy vzít danou adresu a rozdistribuovat ji příslušné komponentě. Nicméně sběrnice tohoto systému má 2 zvláštnosti: Správa virtuální paměti a přímý zápis do vyrovnávací paměti instrukcí.

#### 4.1.1 Virtuální paměť

Jak bylo naznačeno, *PlayStation* je do jisté míry zjednodušená verze deskového počítače a existují určité artefakty, které jsou pozůstatky plné funkčnosti deskového počítače. Jedním z takovýchto artefaktů je virtuální paměť. I přesto, že procesor má podporu *Translation Lookaside Bufferu (TLB)* pro efektivní správu virtuální paměti, *PlayStation* jej vůbec nevyužívá a všechny přístupy do paměti jsou v zásadě absolutní. To, co zůstalo z virtualizace paměti, je maskování všech adres které přijdou na sběrnici a na základě velikosti zapisovaných či čtených dat probíhá maskování odlišně. Nejdříve se u každého přístupu zahodí vrchní 3 bity. Poté u půl-slova (16 bitů) se zahodí spodní 1 bit a u slova (32 bitů) se zahodí spodní 2 bity. Zahodení spodních bitů souvisí se zarovnáním do paměti<sup>2</sup>.

<sup>1</sup>Paměťová mapa PlayStation[4] <https://psx-spx.consoledev.net/iomap/>

<sup>2</sup>Přístup do paměti[4] <https://psx-spx.consoledev.net/memorymap/>

#### 4.1.2 Vyrovnávací paměť instrukcí

Druhá zvláštnost, na kterou je třeba myslet při distribuci čtení a zápisu, je vyrovnávací paměť instrukcí uvnitř *CPU*. Tato vyrovnávací paměť slouží pro rychlé získávání instrukcí z hlavní paměti a teoreticky by nikdy neměla nastat situace, kdy ve vyrovnávací paměti bude něco jiného, než co je obsaženo v hlavní paměti. Ovšem to není u tohoto systému vždy pravda.

*CPU* má speciální stav, který se dá programaticky nastavit a který takzvaně izoluje vyrovnávací paměť instrukcí. Pokud *CPU* se nachází v tomto stavu, pak každé čtení za účelem získat instrukci (*fetch* fáze *CPU*) nikdy nebude putovat do hlavní paměti, ale do této vyrovnávací paměti a **každý** zápis bude modifikovat vyrovnávací paměť místo hlavní paměti. *CPU* tedy jinými slovy poskytuje přímý přístup do této vyrovnávací paměti. Pokud toto není správně ošetřeno a simulováno, *PlayStation* nebude schopen nastartovat<sup>3</sup>.

#### 4.1.3 Časování

Každá hardwarová komponenta pracuje v reálném čase paralelně a nezávisle. Situaci dále zhoršuje fakt, že každá komponenta má odlišnou frekvenci hodin<sup>4</sup>.

- **CPU** - 33.868800 MHz
- **GPU** - 53.222400 MHz
- **SPU** - 44.100 KHz
- **DotClock Timer** - 4.980705 MHz (Průměr)
- **HBlank Timer** - 9923 Hz (NTSC), 9943 Hz (PAL)
- **System Clock/8 Timer** - 4.233600 MHz

Tento problém je řešen tak, že emulátor simuluje každou komponentu sekvenčně a zvlášt v malých kvantech. Množství hodinových cyklů alokovaných pro danou komponentu reflekтуje předchozí list hodnot. Tato simulace může způsobit časovou dilataci mezi jednotlivými komponentami a vést k narušení jejich synchronizace. Při volbě časového kvanta ho nesmíme zvolit příliš malé, protože by došlo k nesprávnému zaokrouhlení na základě časovací tabulky a nepřesnost časování by byla větší. Zároveň však kvantum nesmíme zvolit příliš velké, protože by se komponenty nemusely dostat včas ke slovu. Dvě komponenty, pro které je synchronizace nejdůležitější, jsou *CPU* a *GPU*. Přičemž *GPU* je o  $\frac{11}{7}$  rychlejší, zvolil jsem tedy časovou konstantu 301, protože výsledek formule  $301 \times \frac{11}{7} = 473$  je celé číslo a není třeba řešit zlomkovou část časování.

## 4.2 CPU

Pokud je sběrnice nervovým systémem, pak *CPU* je mozkem *PlayStation* systému. Jde o *MIPS R3000A* 32-bitový *RISC* procesor TODO: source. Jeho architektura je založena na *MIPS I* redukované instrukční sadě. *CPU* má celkem 5 stavů, ve kterých se může nacházet a které provádí v nepřetržité smyčce:

<sup>3</sup>Koprocesor 0, registr 12, bit 16[4] <https://psx-spx.consoledev.net/cpuspecifications/#cop0r12-sr-system-status-register-rw>

<sup>4</sup>Časování[4] <https://psx-spx.consoledev.net/timers/>, <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#gpu-timings>

- **Fetch (IF)** fáze - získání následující instrukce z paměti *RAM*.
- **Decode (ID)** fáze - dekódování instrukce a zjištění následující operace.
- **Execute (EX)** fáze - *CPU* provede dekódovanou instrukci (aritmetickou či logickou operaci).
- **Memory Access (MEM)** fáze - pokud instrukce přistupuje do paměti, data jsou pomocí sběrnice přečtena či zapsána do paměti *RAM*.
- **Write Back(WB)** fáze - Výsledek instrukce je zapsán do souboru registrů.

#### 4.2.1 Vnitřní stav *CPU*

Pro ukládání mezivýsledků a pro obecné zpracování logiky, *CPU* obsahuje celkem **32** 32-bitových registrů, plus **2** 32-bitové registry, které jsou specializované pro práci s násobícími a dělícími instrukcemi[3]. *Nultý* registr je speciální tím, že při čtení vrací vždy nulu a jakýkoliv zápis do něj je ignorován.

#### 4.2.2 Zpoždění načítání hodnot

*MIPS R3000A*, jako každý *RISC* procesor, vykazuje zvláštní jev při načítání hodnot do registrů. Pokud se snažíme načíst hodnotu z paměti *RAM* do procesoru, zpoždění způsobené načítáním hodnoty má za následek opožděné nastavení registru na přečtenou hodnotu o jeden procesorový takt[3]. Toto je způsobeno samotným návrhem *RISC* architektury a je nutné tuto situaci ošetřit.

V emulátoru je tento fenomén simuloval pomocí dvou registrových přihrádek, které fungují jako fronta. Kdykoliv *CPU* chce přečíst hodnotu z paměti *RAM*, místo přímého nastavení registru, přečtená hodnota spolu s indexem výsledného registru v prvním taktu je vložena do fronty a po druhém taktu se zmodifikuje registrové pole.

Je nutné také pamatovat na situaci, kdy ve frontě je připravená hodnota, ale mezitím přijde jiná instrukce, která stejný registr modifikuje. V takovém případě je nutné frontu vyčistit, aby se výsledný registr špatně nezmodifikoval.

#### 4.2.3 Zpoždění skoku

Podobně jako opožděné načítání hodnot do registru, design 5 stupňové architektury má na svědomí ještě jeden problém. Tento artefakt se vyskytuje u všech skokových instrukcí a má za následek, že bez ohledu na to, zda-li se skočí nebo ne (v případě podmíněných skoků), následující instrukce po skoku se vždy provede[3].

To je způsobeno tím, že následující instrukce je již načtena a dekódována uvnitř *CPU*. Kompilátory té doby byly dobře seznámeny s tímto fenoménem a ve většině případů vyplnily instrukci za skokem prázdnou instrukcí. Z tohoto důvodu je každý skok v emulátoru zpožděn o jeden takt.

#### 4.2.4 Instrukční sada

Instrukční sada *CPU*, založená na architektuře *MIPS I*, obsahuje relativně malý počet instrukcí. Tato sada zahrnuje **40** základních instrukcí a **29** rozšířených instrukcí, což dohromady činí **69** instrukcí celkem[7]. Tyto instrukce jsou navrženy tak, aby každá z nich

Instructions encoded by opcode field.								
opcode bits 28..26	0	1	2	3	4	5	6	
bits 31..29	000	001	010	011	100	101	110	
0 000	<i>SPECIAL</i> δ	<i>REGIMM</i> δ	J	JAL	BEQ	BNE	BLEZ	BGTZ
1 001	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2 010	<i>COP0</i> δ,π	<i>COP1</i> δ,π	<i>COP2</i> δ,π	<i>COP3</i> δ,π,κ	*	*	*	*
3 011	*	*	*	*	*	*	*	*
4 100	LB	LH	LWL	LW	LBU	LHU	LWR	*
5 101	SB	SH	SWL	SW	*	*	SWR	*
6 110	*	LWC1 π	LWC2 π	LWC3 π,κ	*	*	*	*
7 111	*	SWC1 π	SWC2 π	SWC3 π,κ	*	*	*	*

Instructions encoded by function field when opcode field = SPECIAL.								
function bits 2..0	0	1	2	3	4	5	6	
bits 5..3	000	001	010	011	100	101	110	
0 000	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1 001	JR	JALR	*	*	SYSCALL	BREAK	*	*
2 010	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3 011	MULT	MULTU	DIV	DIVU	*	*	*	*
4 100	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5 101	*	*	SLT	SLTU	*	*	*	*
6 110	*	*	*	*	*	*	*	*
7 111	*	*	*	*	*	*	*	*

Obrázek 4.2: Kompletní mapa všech **69** instrukcí podle specifikace[7].

zabírala jeden procesorový takt, čímž je udržována neustále plněná 5-ti úrovňová linka. Díky této filozofii mají instrukce velmi málo zodpovědností a jsou v podstatě velmi jednoduché.

Aby se předešlo složitému dekódování instrukcí, jak tomu například je u architektury *x64*, kde každá instrukce může mít různou délku, *MIPS I* definuje každou instrukci jako 32-bitové slovo. Horních 6 bitů pak definuje kódování specifické instrukce. I přesto lze každou instrukci rozdělit do zhruba tří tříd [4.3].

#### 4.2.5 Koprocesory

*MIPS R3000A* ve své instrukční sadě má podporu pro celkem 4 různé koprocesory, avšak *PlayStation* využívá pouze 2 z nich<sup>5</sup>. Koprocesorové instrukce jsou všeobecné a lze za ně substituovat jakýkoliv čip, kromě **koprocesoru 1**, který je dedikován pro práci s čísly s plovoucí desetinnou čárkou (není přítomen v *PlayStation* konzoli)[7].

**Koprocesor 0** slouží ke správě výjimek a přerušení. Koprocesor obsahuje nejen informace o typu výjimky nebo tom, kdo způsobil přerušení, ale také logiku pro zpracování výjimky. Procesor při vyhození výjimky uloží současný stav a jeho kontrolní tok je přenesen na rutinu, která se stará o obsluhu výjimky.

**Koprocesor 2** pak zpřístupňuje komponentu *Geometry Transformation Engine (GTE)*, což je hardware specializovaný pro rychlou práci s lineární algebrou. Tento koprocesor pracuje s dvěma základními datovými primitivy: 3D vektory (16/32-bitový atom) a 3x3 matice

<sup>5</sup>CPU specifikace[4] <https://psx-spx.consoledev.net/cpuspecifications/>

opcode	reg src	reg tgt	reg dst	shift	func
opcode	reg src	reg tgt	immediate		
opcode	target				

Obrázek 4.3: *MIPS* má celkem 3 způsoby, jak zakódovat instrukci do 32 bitů.

(16/32-bitový atom). Vektory mohou být interpretovány jako body v prostoru nebo jako barvy a pro každý typ má *GTE* dedikované příkazy. Funkce *GTE* jsou velmi všeobecné, zahrnují rychlé násobení matice s vektorem, normalizaci barev nebo vektoru a dokonce i interpolaci. Tento koprocessor je nepochybně klíčový pro rychlé zpracování geometrie ve hře a rychlé vykreslení 3D scény.

## 4.3 GPU

Grafická jednotka systému je speciálně navržený čip pro hardwarovou podporu rasterizace geometrických primitiv<sup>6</sup>. *GPU* pracuje na frekvenci 53.222400 MHz, což znamená, že je o  $\frac{11}{7}$  rychlejší než frekvence *CPU*. Všechna komunikace s *GPU* probíhá přes čtyři registry:

- **GP0** (pouze zápis) - registr pro odesílání rasterizačních příkazů a posílání geometrických dat. Různé grafické příkazy mohou mít různou délku (počet zapsaných 32-bitových slov).
- **GP1** (pouze zápis) - registr pro modifikaci stavu *GPU* (například: reset, nastavení rasterizačního okna či potvrzení přerušení).
- **GPUREAD** (pouze čtení) - registr pro čtení *VRAM* paměti nebo pro čtení speciálních registrů.
- **GPUSTAT** (pouze čtení) - registr pro čtení celkového stavu *GPU*.

### 4.3.1 VRAM

*GPU* má také vlastní paměť nazývanou *Video RAM* (*VRAM*). Paměť je rozdělena do 512 řádků o 1024 16bitových slovech. Celková kapacita paměti *VRAM* je tedy  $1024 \times 512 \times 2 = 1048576B = 1MiB$ <sup>7</sup>. *VRAM* paměť slouží pouze k ukládání textur, palet pro indexované textury a výsledného *framebufferu*. Jakákoli data o kreslených primitivech (pozice vrcholů či texturovacích souřadnicích) musí být předána přes registr **GP0**. I přesto, že *GPU* má různé módy týkající se barevné hloubky (24 bitů a 15 bitů), výsledný framebuffer musí mít formát 15bitové barvy.

<sup>6</sup>GPU specifikace[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/>

<sup>7</sup>GPU VRAM[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#gpu-video-memory-vram>

### 4.3.2 Geometrická primitiva

*GPU* dokáže vykreslit 3 geometrická primitiva:

- Osově zarovnaný obdélník
- Čára/Polyčára
- Trojúhelník/Čtyřúhelník

Všechna tato primitiva lze kreslit pomocí **GP0** registru<sup>8</sup>, přičemž je nutné zapsat správný počet argumentů do tohoto registru. Počet argumentů daného primitiva závisí především na různých vlastnostech daného primitiva. Ačkoliv ne všechny primitiva mohou mít všechny vlastnosti, *GPU* dokáže rasterizovat jednotlivá primitiva následujícími způsoby:

- **Texturovací koordináty** - Primitivum má k sobě přiřazenou texturu a každý vrchol má asociované texturovací koordináty.
- **Průhlednost** - Na základě předchozího obsahu ve *VRAM* paměti, *GPU* dokáže mírovat barvy ve čtyřech různých módech<sup>9</sup>:
  - *half-each* -  $result = \frac{background+source}{2}$
  - *additive* -  $result = background + source$
  - *subtractive* -  $result = background - source$
  - *additive* -  $result = background + \frac{source}{4}$
- **Gouraudovo stínování** - Tento atribut je svým názvem trochu zavádějící, neboť gouraudovo stínování souvisí spíše s výpočtem osvětlení. V tomto případě ovšem jde pouze zapnutí interpolace atributů mezi vrcholy primitiva.

### 4.3.3 Barvy

Jak bylo zmíněno, *GPU* dokáže pracovat s 24-bitovými barvami, které obsahují 3 základní barevné komponenty (červená, zelená a modrá). Každému z těchto komponent je alokováno 8 bitů a poté dokáže pracovat s 15-bitovými barvami, kde každé komponentě je přiřazeno pouze 5 bitů.

Ovšem paměť *VRAM* je rozdělena do 16-bitových slov. Je tedy nutné tyto barevné formáty správně mapovat do výsledné 16-bitové barvy. 15-bitová barva se pouze zkopíruje do paměti *VRAM*, přičemž 24-bitová barva je za pomocí techniky *dithering*<sup>10</sup> rozdistribuovana do kreslícího okolí. Paměť *VRAM* v každém fragmentu také ukládá extra 1 bit, který figuruje jako kreslící maska. Pokud je nejvyšší bit v 16-bitové barvě nastaven na 1, pak tento fragment bude ignorován a nic se do něj nevykreslí.

<sup>8</sup>GPU GP0 registr[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#gpu-other-commands>

<sup>9</sup>GPU Semi-transparency[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#semi-transparency>

<sup>10</sup>GPU dithering[4] <https://psx-spx.consoledev.net/graphicsprocessingunitgpu/#24bit-rgb-to-15bit-rgb-dithering-enabled-in-texpage-attribute>

#### 4.3.4 Rasterizace primitiv

U každého ze 3 geometrických primitiv, které *GPU* dokáže vykreslit, je nutno zvolit správný algoritmus pro jeho rasterizaci. U rasterizace osově zarovnaného obdélníku stačí zjistit maximum a minimum ve 2D prostoru. Tento omezený prostor může být následně vyplněn.

Rasterizace čáry vyžaduje sofistikovanější přístup, a to **Digital Differential Analyzer (DDA) algoritmus**<sup>11</sup>, který na základě výpočtu sklonu čáry dokáže vyplnit fragmenty mezi dvěma body ve 2D prostoru.

Trojúhelníková rasterizace se řadí mezi vyplňovací rasterizační algoritmy. Pro jeho implementaci jsem zvolil efektivní **Pinedův algoritmus**[6]. Jeho podstata závisí na rozdelení jednotlivých hran trojúhelníku na poloroviny a u každého fragmentu zjišťovat jeho polohu v závislosti na všech polorovinách daného trojúhelníku.

#### 4.3.5 Zvýšení rozlišení

Zvýšení rozlišení se týká především *GPU* a rasterizace jednotlivých primitiv. *GPU* ve svém interním stavu ukládá mimo jiné meze *framebufferu*. Tyto meze pak figuruji při rasterizaci, kde jakýkoliv pokus kreslit mimo tyto meze bude ignorován. Zároveň tyto meze určují, odkud ve *VRAM* paměti se budou brát data pro zasílání do *CRT* monitoru.

Pro tyto účely je nutné spravovat zvláštní *framebuffer*, který podle nastavení bude mít násobek velikosti současného reálného *framebufferu* uvnitř *GPU*. Při každé modifikaci mezí reálného *framebufferu* je nutné zvláštní *framebuffer* zničit, zrekonstruovat a znova spočítat jeho velikost.

Rasterizaci primitiv je pak nutné zachytit a správně přepočítat jejich pozice, stejně jako upravit jejich omezující vlastnosti. Největší potíž však bude při adresaci textur a jejich palet. Při každém zápisu textury a indexu barvy je nutné koordináty přemapovat do zvláštního *framebufferu*.

### 4.4 DMA

Jelikož *CPU* má hodinovou rychlosť 33.8688 MHz, jakýkoliv přenos dat je nesmírně pomalý. Tento fakt je pouze umocněn v situaci, kdy program chce přenést data z paměti *RAM* do jiné hardwarové komponenty. Pokud bychom měli jednoduchou smyčku pro kopírování dat s prokládanou inkrementací indexu do paměti (abychom vyplnili zpoždění čtení z paměti *RAM*), dostaneme 4 instrukce pro kopírování a 3 instrukce pro správu cyklu [4.4].

```
.loop:           ; loop label
    LW r1 [r2]      ; Load 32-bit value into r1 from address r2
    ADDIU r2 r2 4    ; Increment r2 by 4 to move to the next 32-bit read value
    SW r1 [r3]      ; Store 32-bit value back to ram to address r3
    ADDIU r3 r3 4    ; Increment r3 by 4 to move to the next 32-bit write value
    ADDI r4 -1       ; Update remaining words to copy
    BNE r4 r0 .loop   ; If there are still things to copy, jump back to loop
    NOP             ; Filler, which will get executed regardless if jump is made or not
```

Obrázek 4.4: Pomocí 7 instrukcí a správným prokládáním lze docílit maximální rychlosti při kopírování dat. Každý cyklus přenese 4 byty.

Z teoretického hlediska to znamená, že rychlosť přenosu činí  $\frac{33.8688}{4+3} \times 4 = 19.3536 MB/s$ . Kvůli této nevýhodě obsahuje *PlayStation* komponentu *Direct Memory Access (DMA)*,

<sup>11</sup>DDA algoritmus [https://en.wikipedia.org/wiki/Digital\\_differential\\_analyzer\\_\(graphics\\_algorithm\)](https://en.wikipedia.org/wiki/Digital_differential_analyzer_(graphics_algorithm))

která slouží výhradně k velmi rychlému přenosu dat mezi pamětí *RAM* a hardwarovými komponentami. *DMA* má celkem 7 různých kanálů, přičemž každý kanál specifikuje komunikující komponenty<sup>12</sup>.

- **0 - MDECIN** - Z *RAM* do *MDEC* vstupu
- **1 - MDECOUT** - Z *MDEC* výstupu do *RAM*
- **2 - GPU** - Mezi *RAM* a *GPU*
- **3 - CDROM** - Z *CD-ROM* do *RAM*
- **4 - SPU** - Mezi *RAM* a *SPU*
- **5 - PIO** - Mezi *RAM* a *Expansion Port*
- **6 - OTC** - Mezi *RAM* a *GPU Ordering Table*

*DMA* také poskytuje celkem 3 různé módy přenosu:

- **Word Copy** - Jde o rychlý přenos lineární sekvence 32-bitových hodnot. Maximálně se může přenést až 65536 32-bitových hodnot.
- **Block Copy** - Přenáší se bloky o uživatelem definované délce. Provedení přenosu také závisí na připravenosti komponenty.
- **Linked List Copy** - Přenášená data se dělí na hlavičku a tělo. Hlavička obsahuje délku těla a adresu další hlavičky. Celá datová struktura je pak řetězec hlaviček a těl.

Každý mód přenosu poskytuje velmi rychlé přenosy, protože *DMA* využívá *DRAM Hyper Page* mód, což umožňuje *DMA* přistupovat k *DRAM* řádkům v jednom procesorovém cyklu. Tento přístup má minimální režii, která způsobí, že pro každých 17 cyklů se přečte 16 32-bitových slov.

Při *DMA* přenosu má *CPU* přísná pravidla ohledně přístupu do paměti. Pokud probíhá přenos, *CPU* může přistupovat k vyrovnávacím pamětem a ke svým dvěma koprosorům. Jakmile se *CPU* pokusí přistoupit do paměti, je jeho chod pozastaven, dokud *DMA* přenos není dokončen.

Díky tomuto faktu lze v emulátoru jakoukoliv *DMA* synchronizaci obejít tím, že celý přenos se provede najednou a *CPU* je pozastaveno. Výjimkou je *MDEC*, kde tato komponenta indikuje připravenost přenosu dat.

## 4.5 Ovladač přerušení/výjimek

Aby *CPU* mělo přehled o různých událostech, jako je například dokončení *DMA* přenosu, *PlayStation* má 2 úzce svázané hardwarové komponenty: *Ovladač přerušení* a *Ovladač výjimek*. *Ovladač přerušení* je propojen v podstatě se všemi ostatními komponentami, od kterých je schopen přijímat požadavky na přerušení. Tento čip také obsahuje stavový registr, u kterého lze zjistit, kdo přerušení způsobil.

---

<sup>12</sup>DMA[4] <https://psx-spx.consoledev.net/dmachannels/>

To ale není dostatečné pro pozastavení chodu procesoru. *Ovladač přerušení* je napojen na *koprocesor 0 CPU*, tedy *Ovladač výjimek*, přičemž přerušení je pouze zvláštní typ výjimky.

*BIOS* následně disponuje předem definovanými vektory adres, na jejichž základě se rozhoduje, kam předelovat řízení procesoru a následně zpracovat výjimku.

## 4.6 Časovače

*PlayStation* má 3 různé druhy časových zdrojů, plus jeden časový zdroj pro *CPU*. Jelikož se tyto 3 hardwarové složky příliš neliší, využil jsem *STL* knihovnu *C++* k instanciaci každého ze tří zdrojů.

*První* zdroj se nazývá **Dot Clock**. Tento zdroj souvisí s renderovacím módem *GPU* komponenty. *Dot* v tomto kontextu reprezentuje jeden vykreslený fragment (tzv. ne pixel na obrazovce. Fragment může mít větší či menší velikost než pixel.) a tento zdroj tedy počítá počet vykreslených fragmentů, přičemž rychlosť závisí na vertikálním rozlišení *GPU*.

*Druhý* zdroj také souvisí s *GPU* a jeho název je **Horizontal Blank Clock**. Pokaždé, když *GPU* dokončí kreslení jednoho řádku *Framebufferu*, tento zdroj je inkrementován o jedničku. Rychlosť závisí na regionu konzole a verzi *BIOSu (NTSC/PAL)*.

*Třetí* zdroj je **System Clock**, který odráží zdroj hodin *CPU*, ale je o osminu zpomalený.

Každý z těchto časovačů má také schopnost vyvolat přerušení, pokud dosáhne určité hodnoty. Všechny tyto časovače, kromě svých specifických zdrojů, mají také zdroj hodin *CPU* a jsou schopny svůj zdroj měnit<sup>13</sup>.

## 4.7 MDEC

Tento hardwarový čip je jedním z hlavních komponent, která učinila *PlayStation* velmi populární konzolí. Jelikož se *Sony* rozhodlo použít *CD* jako hlavní úložiště pro hry, každá hra měla v té době velký prostor. V porovnání se svým konkurentem, *Nintendo 64*, který mohl ukládat maximálně *64 MB*, *PlayStation* disponoval až *600 MB*, přičemž hra mohla obsahovat i více disků a hráč mohl během hry měnit disky podle postupu ve hře.

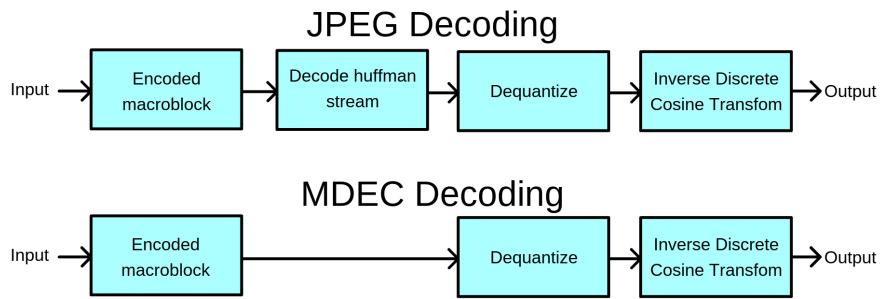
Takový prostor bylo třeba využít a *Sony* se rozhodlo pro video. *MDEC* je čip speciálně věnovaný dekódování speciálního formátu videa. Hlavní myšlenkou tohoto formátu byl *JPEG*. *JPEG* používá několik kompresních technik, které efektivně kombinuje dohromady. Obrázek je rozdělen na *makrobloky* o velikosti *8x8* nebo *16x16* pixelů. Tyto bloky jsou následně převedeny do *YCbCr* barevného formátu a *CbCr* složky jsou zploštěny na polovinu. Toto je možné, protože lidské oko je citlivější na intenzitu světla než na samotnou diferenci barev. Poté je provedena *Discrete Cosine Transform (DCT)*, frekvenční analýza a přebytečné frekvence jsou odstraněny pomocí kvantizační tabulky. Výsledný makroblok je přeorganizován *Zig Zag* vzorem a zakódován pomocí *Run-Length* kódování. *JPEG* pak využívá *Huffmanova* kódování pro vytvoření výsledného datového toku. Dekódování probíhá velmi podobným způsobem, ale obráceným směrem<sup>14</sup>.

Formát snímku videa v *PlayStationu* je velice podobný *JPEGu*, s výjimkou toho, že *PlayStation* nepoužívá *Huffmanovo* kódování.

Při přesunu dat, *MDEC* má speciální příznak pomocí kterého indikuje stav dekódování (připravenost vstupu a výstupu).

<sup>13</sup>Časovače[4] <https://psx-spx.consoledev.net/timers/>

<sup>14</sup>MDEC[4] <https://psx-spx.consoledev.net/macroblockdecodermdec/>



Obrázek 4.5: *MDEC* má velmi podobnou strukturu jako *JPEG* až na vynechaný krok Huffmanova dekódování.

## 4.8 CD-ROM mechanika

*Sony* se rozhodlo použít pro herní úložiště *Compact Disc Extended Architecture (CD-XA)* a standard ISO 9660. *BIOS* se stará o analýzu ISO 9660 souborového systému, *PlayStation* obsahuje speciální *CD-ROM* mechaniku pro čtení *CD*.

*CPU* může komunikovat s *CD-ROM* mechanikou jednak přes registry a příkazy, ale *CD-ROM* má také tři datové fronty, pomocí nichž se přenášejí data a dotazy na přerušení. Interně *CD-ROM* disponuje celkem 37 příkazy, pomocí kterých lze manipulovat s čtecím motorem.

Pro správný přístup k *CD* je nutné správně adresovat toto médium. Jelikož je *CD-ROM* spíše chápán jako úložiště pro audio či video média, disk je adresován pomocí stop. Každá stopa se dá reprezentovat indexem nebo formátem *minuty (MM):sekundy (SS):zlomky (FF)*. Každý disk má maximálně 74 minut, přičemž v každé minutě je 60 sekund a každá sekunda obsahuje 75 zlomků. S touto znalostí se pak tento formát dá převést na *Logical Block Addressing (LBA)*, což je schopnost adresovat *CD* jako lineární paměť bytů.

Hry také měly možnost být prodávány ne na jednom, ale na několika discích. *CD-ROM* má pak schopnost výměny disku bez resetu konzole. Pomocí vypnutí motoru lze *CD-ROM* mechaniku pozastavit, otevřít poklop, vyměnit disk a program po uzavření poklopů může bezproblémově pokračovat<sup>15</sup>.

## 4.9 Periférie

Ačkoliv samotná konzole má vnitřně několik typů periférií, jako jsou I/O a debugovací porty, veřejně dostupná verze konzole má celkem 4 porty, se kterými běžný uživatel může interagovat. Z těchto portů slouží 2 na vsunutí *MemoryCard*, což figurovalo jako malé úložiště pro uchování postupu ve hrách. Zbývající 2 porty pak fungují jako vstupy pro herní ovladače, kde existují celkem 3 podporované typy: *Digital*, *Analog* a *Mouse*. Nicméně drtivá většina celé *PlayStation* knihovny používá pouze *Digital* verzi ovladače.

Hry mohou individuálně číst z těchto portů a umožňují tak uživateli s konzolí interagovat. Veškerá komunikace mezi *CPU* a perifériemi probíhá pomocí *Serial I/O (SIO)*<sup>16</sup>.

<sup>15</sup>CD-ROM mechanika[4] <https://psx-spx.consoledev.net/cdromdrive/>

<sup>16</sup>Ovladače a paměťové karty[4] <https://psx-spx.consoledev.net/controllersandmemorycards/>

# Kapitola 5

## BIOS

### 5.1 Odrazový bod

Aby člověk byl schopen vytvořit emulátor jakéhokoliv systému, je vždy potřeba určitý odrazový bod, od kterého lze začít implementovat emulátor a který současně slouží i jako test korektnosti jeho fungování. Ve většině případů jde o zaváděcí program, který inicializuje daný systém. V případě *PlayStationu* se jedná o *Basic Input/Output System (BIOS)*, který je zabudován do *ROM* paměti každé *PlayStation* konzole. Existuje několik verzí *BIOSu*. Hlavní dělení probíhá podle regionálních/verzních základních desek, kde se rozlišují tři hlavní verze *BIOSu*: Americký (*SCEA*), Japonský (*SCEI*) a Evropský (*SCCE*). Sony také vytvořilo modifikované verze *BIOSu*, které používalo v konzolích dalších generací pro hladší emulaci *PlayStation* systému (například systém *PlayStation Portable* obsahuje upravený *BIOS*, který přeskakuje bootovací úvodní animaci).

Tento *BIOS* lze využít pro strukturovanou implementaci celého emulátoru, kde je nejprve vytvořena paměťová struktura emulátoru podle paměťové mapy a poté jsou postupně spouštěny instrukce *BIOSu*. Jakmile emulátor narazí na instrukci, která zahrnuje funkcionality či přístup do ještě neimplementované hardware komponenty, emulátor signalizuje chybu a chybějící funkcionality je třeba buď plně implementovat, nebo ji utlumit, v závislosti na její závažnosti a schopnosti systému fungovat bez ní.

### 5.2 Funkce

Ačkoliv se tomuto zaváděcímu programu v *PlayStation* komunitě říká *BIOS*, jedná se v podstatě o velmi odlehčený operační systém, který poskytuje systémová volání pro usnadněný přístup k hardwaru.

Uživatel mohl spustit *PlayStation* konzoli i bez vložené hry do *CD-ROM* čtečky a byl uvítán úvodní obrazovkou *BIOSu*, tj. *shellem*. V tomto menu měl uživatel možnost spravovat, kopírovat a mazat uložený postup her, pokud byla do konzole připojena speciální paměťová karta a také uživatel mohl v tomto menu přehrát *Audio CD*.

Hlavní funkcionality *BIOS* však poskytuje prostřednictvím speciálních instrukcí systémových volání nebo speciálních tabulek rutin umístěných na začátku paměti *RAM*. Skrze toto *API BIOS* nabízí řadu funkcí, jako je například přístup k souborovému systému *CD-*

*ROM*, správa paměti, výpis ladění, ale také zpracování výjimek, manipulace s řetězci, přístup k *GPU*, *SPU*, *GTE* a mnoho dalších<sup>1</sup>.

## 5.3 Fáze BIOSu

### 5.3.1 Zaváděcí fáze

Ačkoliv se jednotlivé verze *BIOSu* liší ve své funkcionalitě, všechny následují velmi podobný zaváděcí proces. Při zapnutí/resetu konzole se *Program Counter* procesoru nastaví na hodnotu **0xBFC00000**, což je začátek adresy, kde je uložen *BIOS*. V tomto bodě je uložena resetovací logika. V prvé řadě jsou registry *CPU* vyčištěny a poté začne inicializovat hardware.

Nejdříve se nastaví registry *paměťového ovladače*. Tento kus hardwaru je diskutabilně do jisté míry zbytkem z reálného počítače, obsahující informace o velikostech jednotlivých pamětí (*RAM*, *BIOS*, *Scratchpad*, ...).

Dále se odizoluje *vyrovnávací paměť instrukcí*, což způsobí, že všechny zápisy se převedou místo na sběrnici do této *vyrovnávací paměti*. *BIOS* poté tuto paměť vyplní a vyčistí nulami.

V další fázi *BIOS* přistupuje k *ovladači vyrovnávací paměti* a zapne instrukční a datovou vyrovnávací paměť. Tato komponenta slouží jako globální vypínač jednotlivých vyrovnávacích pamětí (*d-cache*, *i-cache*, *scratchpad*), ovšem podobně jako *paměťový ovladač*, jde spíše o formalitu, neboť do tohoto ovladače se dále už nepřistupuje.

*BIOS* pak zresetuje **koprocesor 0** (ovladač výjimek) tak, že všechny jeho registry nastaví na nulu a utlumí všechny kanály *SPU*.

Pokud *BIOS* běžel na vývojářské verzi konzole, programátor mohl využít *Expansion Port (PIO)*, což do jisté míry je *GPIO* použitelné pro debugování vývoje hry. Aby *PIO* mohlo být použito, zařízení na druhé straně muselo zaslat speciální ASCII řetězec: *"Licensed by Sony Computer Entertainment Inc."* kvůli verifikaci.

### 5.3.2 Jádro BIOSu

Po inicializačním resetu hardware je do paměti zkopirován obraz jádra a spuštěn. Jádro zpřístupní svou funkcionalitu tím, že vyplní speciální rutinní tabulky na začátku paměti *RAM*, spustí *shell* a zobrazí úvodní animaci.

Po dokončení této animace je zkontrolována *CD-ROM* mechanika, zda-li je její poklop uzavřen a zda-li mechanika obsahuje *CD*. Pokud jsou tyto podmínky splněny, *BIOS* začne analyzovat souborový systém *CD* a hledat hlavní spustitelný soubor. Pokud je přítomen, je posléze spuštěn.

Pokud v *CD-ROM* mechanice nic není, *BIOS* spustí *shell* a dále se již s ničím nadále nezabývá.

---

<sup>1</sup>PCSX ReARMed HLE BIOS [https://github.com/notaz/pcsx\\_rearmed/blob/master/libpcsxcore/psxbios.c](https://github.com/notaz/pcsx_rearmed/blob/master/libpcsxcore/psxbios.c)



Obrázek 5.1: Úvodní obrazovka indikující správnou inicializi systému.

# Kapitola 6

## Závěr

V tomto projektu neočekávám, že můj emulátor na konci bude mít 100% kompatibilitu s celou *PlayStation* knihovnou. Rozhodně budu rád za to, pokud bude jakákoli jedna hra funkční a spustitelná. V tomto projektu jde hlavně o to prozkoumat komerční a velmi populární systém a pokusit se zpříjemnit jeho používání pro komunitu herních nadšenců milující staré hry, ať už kvůli zvědavosti, nebo kvůli nostalgií.

Implementace takto složitého systému vyžaduje opominutí určitých specifických chování hardwaru, přeskočení logiky fungování komponentu, nebo dokonce přeskočení celistvé komponenty. Čili nebudu zastírat, že tu a tam chybí cihla či trám. Tato neduha je způsobena tím, že mohu čerpat pouze z dokumentů, které jsou výsledkem mnohaleté práce reverzního inženýrsví mnoha lidí a ačkoliv se jedná o monumentální úsilí, nakonec nikdo nemá přístup k originálnímu hardwarovému návrhu *PlayStation* konzole.

# Literatura

- [1] *Sony Computer Entertainment v. Connectix Corp.*, 203 F.3d 596 (9th Cir. 2000) [online]. 2000. Dostupné z: <https://casetext.com/case/sony-computer-entertainment-v-connectix-corp-2>.
- [2] *Nintendo PlayStation prototype - PlayStation* Wikipedia [online]. 2020. Dostupné z: [https://playstation.fandom.com/wiki/Nintendo\\_PlayStation](https://playstation.fandom.com/wiki/Nintendo_PlayStation).
- [3] INTEGRATED DEVICE TECHNOLOGY, I. *IDT R30xx Family Software Reference Manual* [online]. 1994. Dostupné z: <https://hitmen.c02.at/files/docs/psx/3467.pdf>.
- [4] KORTH, M. 'nocash'. *PlayStation Specifications* [online]. 2022. Dostupné z: <https://psx-spx.consoledev.net/>.
- [5] NEWZOO.COM. *Key Numbers*, Archived from <https://newzoo.com/key-numbers/> [online]. 2019. Dostupné z: <https://web.archive.org/web/20190509014637/https://newzoo.com/key-numbers/>.
- [6] PINEDA, J. *A Parallel Algorithm for Polygon Rasterization* [online]. Apollo Computer Inc., 1988. Dostupné z: <https://www.cs.drexel.edu/~deb39/Classes/Papers/comp175-06-pineda.pdf>.
- [7] PRICE, C. *MIPS IV Instruction Set* [online]. 1995. Dostupné z: <https://www.cs.cmu.edu/afs/cs/academic/class/15740-f97/public/doc/mips-isa.pdf>.
- [8] SALVADOR, P. *Survey of the Video Game Reissue Market in the United States (1.1)* [online]. 2023. Dostupné z: <https://doi.org/10.5281/zenodo.8161056>.